File I/O - Writing Data Files

Writing to data files is very similar to reading data files There are still just three basic steps:

1. Open the file
2. *Write the data*
3. Close the file

The way in which we open the file will change slightly (the mode will be `"w"` or `"a"`); the biggest difference is that we will use the `fprintf()` function to write data to a file much as we used `printf()` to write text or data to the screen.

Opening the File

Recall, in the notes on reading data from files, we said that there are three "modes" in which a file could be opened, but that we would only need the read (`"r"`) mode for reading data. As a reminder, the three modes are:

1. write ("w") - write to the file, destroying any existing file content
2. read ("r") - read from the file
3. append ("a") - write to the file, appending new data to the end of any existing file content

To write to a file, we will use either the write (`"w"`) or append (`"a"`) mode. There is an **important difference** between these two modes: if you open a file in write mode, you will destroy the previous contents of the file, whereas if you open a file in append mode, the previous contents of the file are retained, and any new output will be appended to the file.

Example: Open the file `grade_report.txt` for writing. It is okay if the previous contents of the file are lost.

```
FILE *fp;
fp = fopen("grade_report.txt", "w");
```

Example: I've written a program to compute statistics from daily observations of some bacteria cultures. The lab equipment produces a file of data each day, I will run my program to process the data, and the output of my program needs to be written to the file `daily_stats.txt`. It is important that all the reports be retained, so the file should be opened in append mode.

```
FILE *fp;
fp = fopen("daily_stats.txt", "a");
```

Writing the Data

We will use the `fprintf()` function to write output to a file. It works just like `printf()`, except its first argument is the file pointer of the output file.

When reading data, it is important to know how the data file is formatted; when writing data, you get to choose the format of your output. For example, if I have a program that works with height and weight for a number of study participants, I can choose to write an individuals height and weight as floats or ints, separated by space, tab, or comma, etc.

Example: Write the `fprintf()` function to write height and weight as comma-separated floats. Assume the file pointer name of the output file is out_fp.

```
fprintf( out_fp, "%f,%f\n", height[i], weight[i] );
```

Closing the File

This is exactly the same as for reading data files:

```
fclose( fp );
```

Example: My program reads a file containing all my students' homework, project, and exam scores and computes each student's final, weighted score. Write the C code to write weighted scores to the file `scores.dat`. Assume the constant `NUMSTUDENTS` is the number of students in the class and that the scores are stored in the array `float scores[NUMSTUDENTS];`.

```
FILE *fp;

fp = fopen("scores.dat", "w");

for (i = 0; i < NUMSTUDENTS; i++) {
    fprintf(fp, "%f\n", scores[i]);

fclose( fp );
```

Example: Re-write the `printf()` functions from the example program so that the output is written to the file `grade_report.txt`. It's okay to overwrite the previous content of the file.

```
     FILE *out_fp;
     out_fp = fopen( "grade_report.txt", "w" );

     fprintf(out_fp, "\n");
     fprintf(out_fp, "Grade Statistics\n\n");
     fprintf(out_fp, "\tExam 1\tmedian = %.2f\tmean=%.2f\tmin=%.2f\tmax=%.2f\n", median1, mean1, min1, 
max1);
     fprintf(out_fp, "\tExam 2\tmedian = %.2f\tmean=%.2f\tmin=%.2f\tmax=%.2f\n", median2, mean2, min2, 
max2);

     fprintf(out_fp, "\n");
     fprintf(out_fp, "Grade Distribution\n\n");
     fprintf(out_fp, "\tExam 1\tA:%d\tB:%d\tC:%d\tD:%d\tF:%d\n", gc1[A], gc1[B], gc1[C], gc1[D], gc1[F]);
     fprintf(out_fp, "\tExam 2\tA:%d\tB:%d\tC:%d\tD:%d\tF:%d\n", gc2[A], gc2[B], gc2[C], gc2[D], gc2[F]);

     fprintf(out_fp, "\n");
     fprintf(out_fp, "Mean delta (ex2 - ex1) is %f\n\n", mean_change);

     fclose( out_fp );
```

File I/O - Detecting Errors

What happens if a program tries to open a file that doesn't exist? Or tries to write a file in a directory it does not have access to? The offending function will return a value that can be used to determine that an error occurred. There are a number of ways to detect errors when reading or writing files, but we will just cover three:

1. Attempting to open a file with `fopen()` that does not exist or that you do not have permission to write to.
2. Reading lines of data that do not match the format statement in `fscanf()`.
3. Failure when writing data to a file with `fprintf()`.

Failure Opening a File

As we've seen, the `fopen()` function returns a file pointer (type `FILE *`). It turns out that if fopen() cannot open the file for one reason or another, it returns the special value `NULL`. Thus, if the return value of `fopen()` is `NULL`, we know an error occurred.

Example: Suppose the file `bad.data` does not exist. The following code will detect that the attempt to open the file for reading failed.

```
FILE *fp;

fp = fopen("bad.data", "r");

if (fp == NULL) {
     printf("Error - could not open input file.\n");
     exit(1);
}
```

This can also be written more compactly as follows:

```
FILE *fp;

if ( (fp = fopen("bad.data", "r")) == NULL ) {
     printf("Error - could not open input file.\n");
     exit(1);
}
```

Note that in the previous example, we also introduced a new function: `exit()`. The `exit()` function causes the program to quit and to return its argument (an integer) to the Linux command shell. Whether the program should exit or not when it encounters a file error depends on how crucial that file is to the operation of the program. If reading the specified file is essential, then the program should definitely print a message and exit if there is an error opening the file.

Important: to use the `exit()` function you must also include `stdlib.h`, i.e. you must have the line "`#include <stdlib.h>`" near the top of your C source file.

It is less common to receive an error when opening a file for writing or appending since the file does not have to exist already. However, we can use the same technique to detect errors in these cases: just check if the return value of `fopen()` is `NULL`.

Failure Reading from a File

A source of errors when reading data is a mis-match between the format string in `fscanf()` and the actual format of the data file. This can occur because the format string is incorrect or because of errors in the data file. The `fscanf()` function returns the number of items matched in the format string; we can compare this to the expected number of matches, and if they differ, there is an error.

Example: Suppose the file `garbled.data` consists of two columns of floating point numbers separated by commas:

```
123.4,17.2
209.5, 5.1
98.3, 2.1
```
etc.

Further, suppose there was an error when the data file was produced that resulted in one line having only a single floating point number and no comma. We would read each line of data using an expression like:

```
fscanf(fp, "%f,%f", &x[i], &y[i]);
```

We expect the `fscanf()` function to return the value 2 since there are two items to be matched in the format string (the two floats). However, when `fscanf()` encounters the garbled line, it will return the value 1 since it will only be able to match the first "`%f`". We can test for this as follows:

```
FILE *fp;
int i;
int ret;

for (i=0; i<numdata; i++) {
     ret = fscanf(fp, "%f,%f", &x[i], &y[i]);
     if ( ret != 2) {
          printf("Error - invalid data line.\n");
          exit(1);
     }
}
```

Again, this can be made slightly more compact as follows:

```
for (i=0; i<numdata; i++) {
     if ( fscanf(fp, %f,%f", &x[i], &y[i]) != 2) {
          printf("Error - invalid data line.\n");
          exit(1);
     }
```

Notice that again I have chosen to exit the program when the error occurs; however, if the error is not critical to the purpose of the program, you may choose to just print a warning message, or to skip the invalid line without exiting.

Failure Writing to a File

Lastly, we mention how to detect an error when writing a file. The `fprintf()` function returns the number of bytes written or a negative number if a major error occurs. This is not as informative as it sounds, because we often do not know *exactly* how many bytes a call to `fprintf()` should produce. For example,

```
fprintf(fp, "%d: %f\n", id, height);
```

will produce a different number of bytes of output depending on the values of id and height. However, we expect our call to `fprintf()` to produce some output, so it is often sufficient to check if the return value is less than or equal to zero; if it is, an error occurred.

Example: My program has patient data for `numpatient` individuals and I want to print each patient's `id` and `weight` to a file. I can check for an error on write as follows:

```
FILE *fp;
int i;
int ret;

for (i=0; i<numpatient; i++) {
     ret = fprintf(fp, "%d: %f\n", id[i], weight[i]);
     if ( ret <= 0 ) {
          printf("Error - failed to write data to output file.\n");
          exit(1);
     }
}
```

Or alternatively,

```
for (i=0; i<numpatient; i++) {
     if ( fprintf(fp, "%d: %f\n", id[i], weight[i]) <= 0 ) {
          printf("Error - failed to write data to output file.\n");
          exit(1);
          }
}
```